



Dipartimento  
di Fisica  
e Astronomia  
Galileo Galilei

1222-2022  
**800**  
ANNI



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

# Sperimentazioni di Fisica I

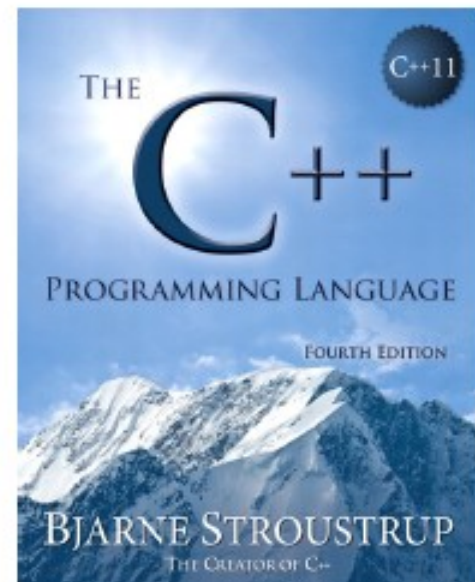
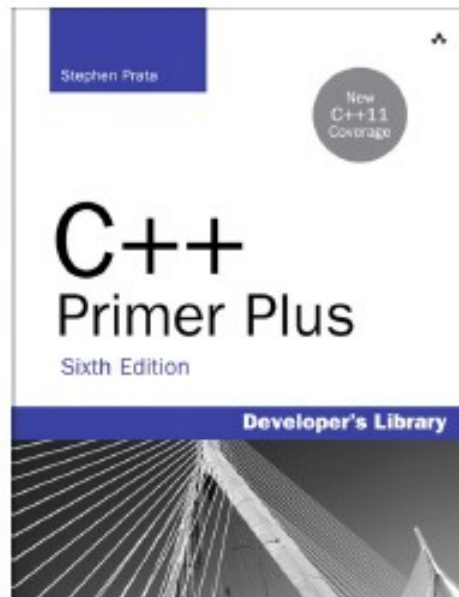
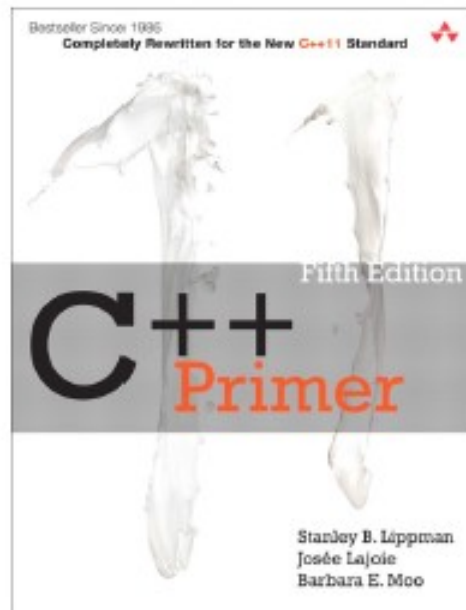
## mod. A – Lezione 6

### **Introduzione al C++ (CAP. 1)**

*Dipartimento di Fisica e Astronomia “G. Galilei”,  
Università degli Studi di Padova*

# Testi di Riferimento

- Libri di testo **consigliati** :
  - ✓ Lippman, Lajoie, Moo, “C++ Primer”, Addison-Wesley, ISBN: 978-0321714114
  - ✓ S. Prata, “C++ Primer Plus”, Addison-Wesley, ISBN: 978-0321776402
- eventuali libri di **consultazione avanzata** :
  - ✓ B. Stroustrup, “The C++ Programming Language”, Addison-Wesley, ISBN: 978-0321563842
  - ✓ Brandolese, Giuliani, “C++2011, Programmazione e caratteristiche dello standard 2011”, Pearson, ISBN:978-88-7192-645-2



# Scrittura di un Programma in C++

- Ogni programma in C++ è costituito da una o più funzioni
- La funzione principale (e irrinunciabile) è chiamata `main`

- Programma minimale

```
int main ()  
{  
    return 0;  
}
```

The code is annotated with four numbered boxes: box 1 is around 'int', box 2 is around 'main', box 3 is around '()', and box 4 is around the opening curly brace '{'.

- Per definire una funzione bisogna specificare 4 elementi chiave :
  1. un tipo di ritorno
  2. il nome della funzione
  3. una lista di parametri (può essere vuota)
  4. il corpo della funzione

# La Funzione main ( )

- È l'elemento fondamentale, **irrinunciabile**

Punto di partenza per ogni programma

- Il tipo di ritorno è `int` (intero)

built-in type del C++

- La lista dei parametri può essere vuota

nelle parentesi ( ), dopo il nome

```
int main()
{
    ...
    return 0;
}
```

no parameters

```
int main(int argc,
         char ** argv)
{
    ...
    return 0;
}
```

C-style format

```
int main(int argc,
         char * argv[])
{
    ...
    return 0;
}
```

C++ format

# Compilazione ed Esecuzione

1. Salvare il programma in un file con estensione `.cc`, `.cxx`, `.cpp`, `.cp`, oppure `.C`
2. Invocare il compilatore da linea di comando (dalla shell)

```
$ g++ program0.cxx
```

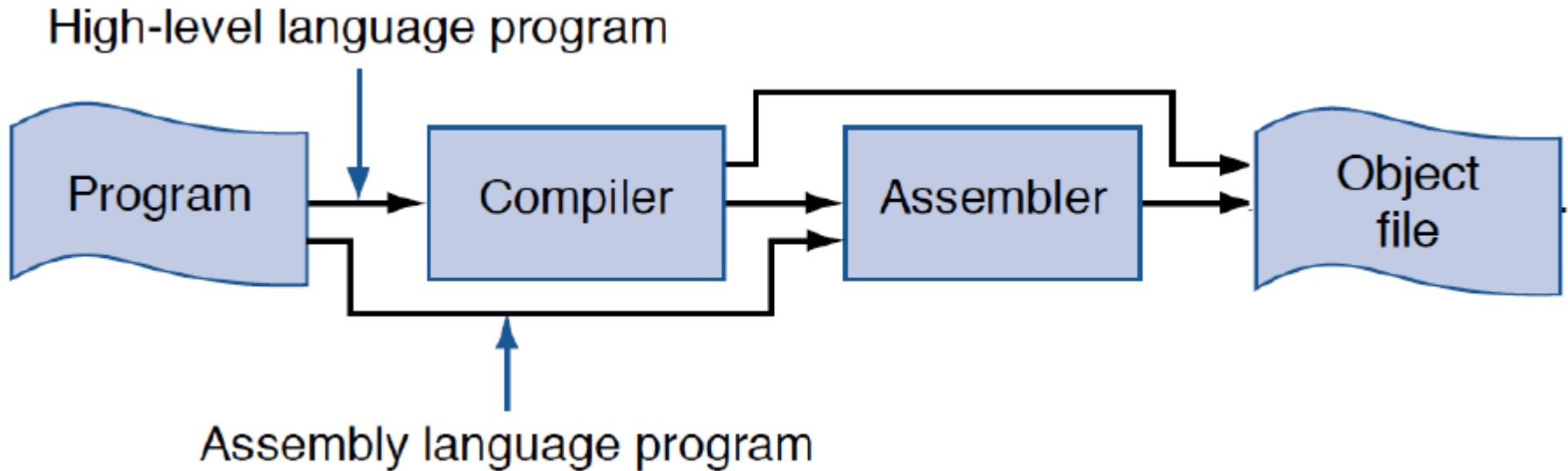
3. Viene creato un file eseguibile di nome `a.out`
4. Eseguire il programma

```
$ ./a.out
```

5. Verificare il valore di ritorno dalla shell

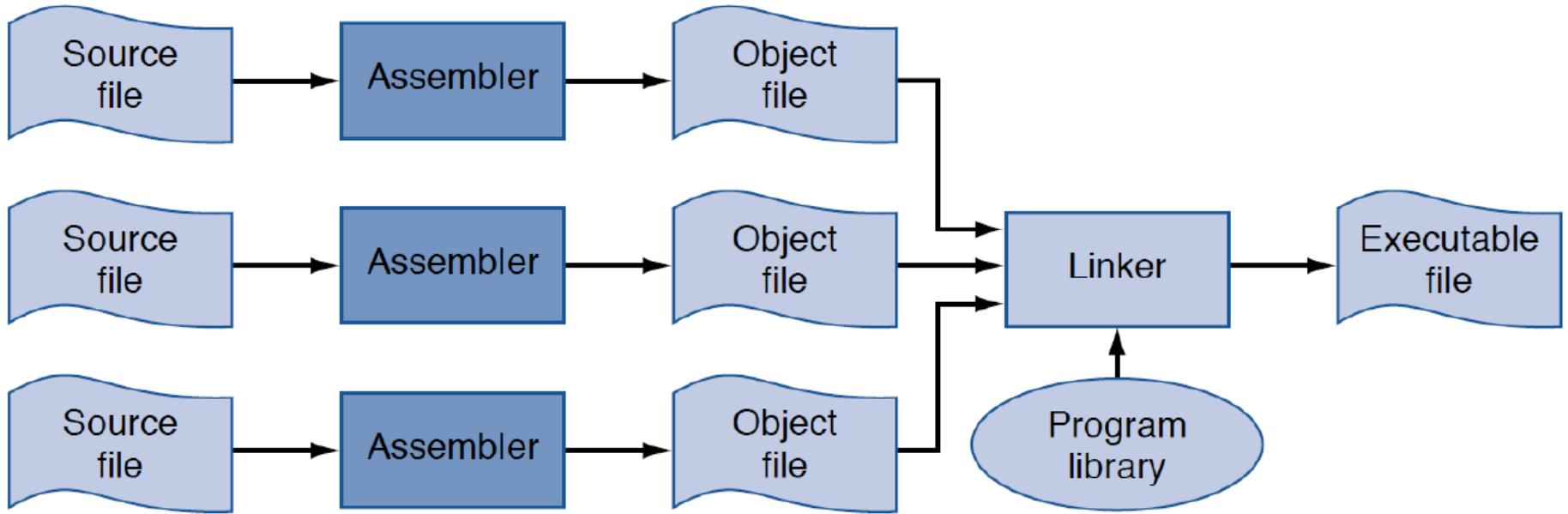
```
$ echo $?  
0
```

# “compilazione” del codice



- Compilazione/Assembly
- Linking
- Esecuzione

# Generalizzazione

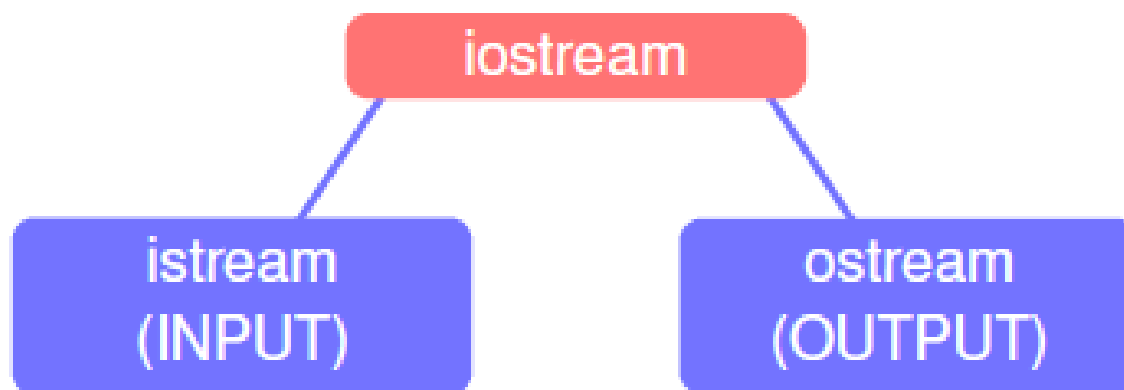


.. e la std library?

.. se ho molti file come gestisco la procedura (Make, Automake, Cmake... suite)

# Input ed Output in C++

- Il C++ non definisce nessuna istruzione specifica di I/O
- Viene fornita una intera libreria (Standard Library) con classi e oggetti specifiche per I/O



- **Stream:** sequenza di caratteri letti o scritti da dispositivi di I/O (tastiera, terminale, file su disco, file in rete)

i caratteri vengono generati/acquisiti sequenzialmente nel tempo



# La Libreria `iostream`

- La libreria definisce 4 oggetti (con relative classi):

- ✓ `istream`

- + `cin` : lettura da standard input `stdin`

- ✓ `ostream`

- + `cout` : scrittura sullo standard output `stdout`

- + `cerr` : scrittura su standard error `stderr`

- + `clog` : scrittura su log (standard logging stream), di default sincronizzato su `cerr`

- in UNIX (LINUX et al.), e WINDOWS vengono gestiti dal sistema operativo

# iostream: esempio di utilizzo

- scriviamo un programma per il calcolo della somma di due numeri interi

```
#include <iostream>
int main()
{
    std::cout << "Inserire due interi: ";

    int v1 = 0, v2 = 0;
    std::cin >> v1 >> v2;

    std::cout << "La somma di " << v1;
    std::cout << " + " << v2 << " = "
              << v1+v2 << std::endl;

    return 0;
}
```


- compiliamo ed eseguiamo il programma

```
$ g++ sum_int.cxx
$
$ ./a.out
Inserire due interi: 3 4
La somma di 3 + 4 = 7
$
$ echo $?
0
```

# #include <iostream>

```
#include <iostream>
```

direttiva del  
preprocessore



```
int main()
{
    std::cout << "Inserire due interi: ";

    int v1 = 0, v2 = 0;
    std::cin >> v1 >> v2;

    std::cout << "La somma di " << v1;
    std::cout << " + " << v2 << " = "
              << v1+v2 << std::endl;

    return 0;
}
```

- ✓ Include un **header file** per poter usare le librerie di I/O
- ✓ ogni programma che voglia usare delle librerie (STL o altre) deve includere gli header files corrispondenti
- ✓ Normalmente **tutti gli header file** sono inclusi **in testa al programma**, fuori dalle funzioni

# ostream: std::cout

```
#include <iostream>
```

```
int main()
```

```
{  
  std::cout << "Inserire due interi: ";
```

```
  int v1 = 0, v2 = 0;  
  std::cin >> v1 >> v2;
```

```
  std::cout << "La somma di " << v1;  
  std::cout << " + " << v2 << " = "  
  << v1+v2 << std::endl;
```

```
  return 0;  
}
```

È un manipolatore:  
invia un carattere  
newline

equivalente a

```
(std::cout << "La somma di") << v1;
```

✓ `std::cout` è un oggetto della classe `ostream`

;

✓ L'operatore `<<` è binario:

✓ a `sx`: oggetto `ostream`

✓ a `dx`: operando del quale stampare il valore

Il primo stampa una stringa, il secondo un intero

# istream: std::cin

```
#include <iostream>

int main()
{
    std::cout << "Inserire due interi: ";

    int v1 = 0, v2 = 0;
    std::cin >> v1 >> v2;

    std::cout << "La somma di " << v1;
    std::cout << " + " << v2 << " = "
              << v1+v2 << std::endl;

    return 0;
}
```

equivalente a

```
(std::cin >> v1) >> v2;
```

- ✓ Definiamo due variabili `v1` e `v2` di tipo `intero` per poter immagazzinare i dati di input
- ✓ `std::cin` è un oggetto della classe `istream`
- ✓ L'operatore `>>` è binario:
  - ✓ a sx: `oggetto istream`
  - ✓ a dx: `oggetto` nel quale immagazzinare l'input
- ✓ Come nell'esempio precedente è possibile combinare una sequenza di input

# La Standard Library `std`

- Il prefisso `std::` indica che `cout` e `cin` sono definiti nel **namespace** chiamato **`std`**
- ✓ Tutti i nomi definiti dalla Standard Library sono nel namespace `std`
- I namespace **permettono di evitare collisioni** tra nomi definiti dall'utilizzatore e nomi di libreria



La dichiarazione `using` permette di accedere ad un oggetto nel namespace senza specificarlo con la solita sintassi

`namespace_name::prefix`

`using std::cin`

`using std::cout`

# using namespace std;

Per ogni oggetto presente in un namespace che vogliamo usare, è necessaria una dichiarazione `using` separata :

```
#include <iostream>
using std::cout;
using std::endl;
using std::cin;

int main() {
    cout << "Dammi un numero:";
    int numero = 0;
    cin >> numero;
    cout << "Numero inserito: ";
    cout << numero << endl;
}
```

È possibile rendere visibile al programma tutti gli oggetti e le classi di un namespace con la dichiarazione

```
using namespace namespace_name;
```

`using namespace std;` per il namespace delle librerie standard del C++

# I Commenti nel Codice

- I commenti migliorano **leggibilità e comprensione** del codice
- Sono **ignorati dal compilatore** e non hanno quindi **nessun effetto sulle prestazioni** dei programmi
- Due sintassi possibili:
  - ✓ **single-line** : iniziano con due caratteri 'slash' (//) e terminano a fine riga. tutti i caratteri dalla destra dei due caratteri // sono ignorati dal compilatore
  - ✓ **paired** : sono racchiusi dai marcatori /\* e \*/

```
// Commento
```

```
int main()  
{  
    return 0;  
}
```

```
int main(int argc,  
          char * argv[])
```

```
{  
    int v = 1; // Un commento  
    return 0;  
}
```

```
/*
```

```
    Un commento su piu' righe  
*/
```

```
int main(int argc,  
          char ** argv)  
{  
    return 0;  
}
```



# Controllo del Flusso del Programma

- Le istruzioni di un programma sono eseguite sequenzialmente
- a volte è necessario alterare il normale flusso di esecuzione
  - ✓ eseguendo blocchi di istruzioni in maniera ripetitiva

L'istruzione `for`

L'istruzione `while`

- ✓ eseguendo porzioni di codice soltanto se determinate condizioni sono verificate

L'istruzione `if`

# Il Ciclo for

- il ciclo `for` permette di **eseguire ripetutamente** una blocco di codice un **numero fissato** (o variabile) di volte

```
#include <iostream>

int main()
{
    int sum = 0;

    for (int val=0; val <= 10; ++val) {
        sum += val;
    }
    std::cout << "Somma da 1 a 10 => "
              << sum << std::endl;
    return 0;
}
```

- ✓ compiliamo ed eseguiamo il programma:

```
$ g++ sum10_for.cxx
$
$ ./a.out
Somma da 1 a 10 => 55
```

# Il Ciclo for

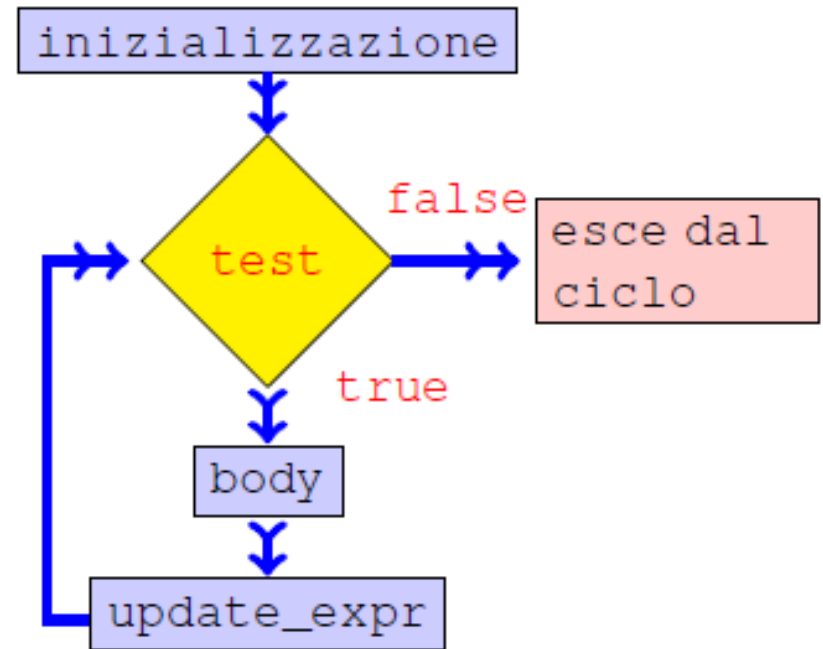
- la sintassi del comando è:

```
for (init; test; update)  
    statement
```

- Il ciclo `for` è composto di due parti:

1. un `header`
2. un `body`

- l'`header` controlla quante volte viene eseguito il `body`.



È composto di tre parti:

1. `inizializzazione`
2. `condizione` da verificare per l'esecuzione del ciclo
3. `espressione` di aggiornamento di fine ciclo

# Il Ciclo while

- l'istruzione `while` esegue ripetutamente un blocco di codice fintanto che una condizione fissata è vera

```
#include <iostream>

int main()
{
    int sum = 0, val = 1;

    while (val <= 10) {
        sum += val;
        ++val;
    }
    std::cout << "Somma da 1 a 10 => "
              << sum << std::endl;
    return 0;
}
```

✓ compiliamo ed eseguiamo il programma:

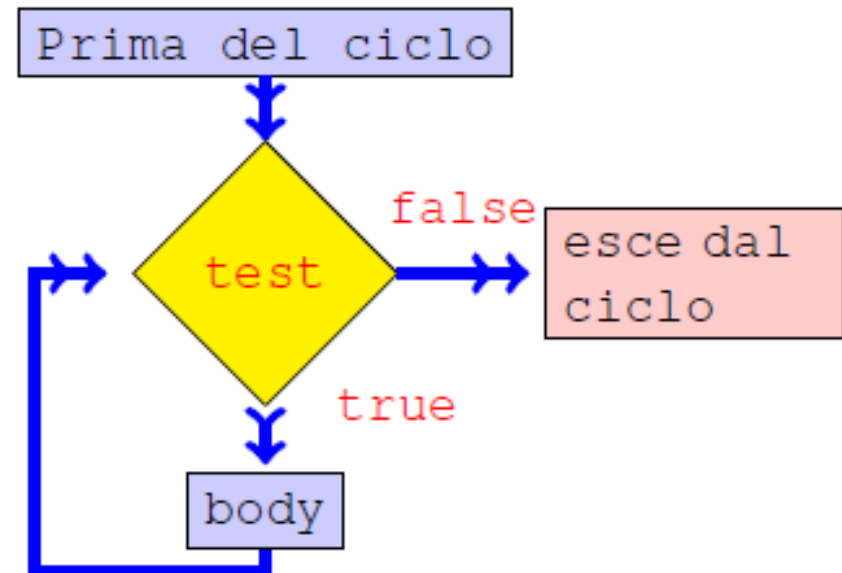
```
$ g++ sum10_while.cxx
$
$ ./a.out
Somma da 1 a 10 => 55
```

# Il Ciclo while

- la sintassi del comando è:

```
while (condition)
    statement
```

- la **condizione** viene controllata ad ogni ciclo, fino a quando è falsa
- **blocco** = sequenza di zero o più istruzioni. Può essere usato dove è richiesto uno statement



✓ `sum += val;` operatore (binario) `+=` composto di assegnazione



✓ `sum = sum + val;`

✓ `++val;` operatore (unario) di incremento  $\rightarrow val = val + 1;$

# Esempio di Utilizzo

```
#include <iostream>

int main()
{
    int sum = 0, value = 0;

    while (std::cin >> value) {
        sum += value;
    }
    std::cout << "The sum is: "
              << sum << std::endl;
    return 0;
}
```

- compiliamo ed eseguiamo il programma

```
$ g++ read_many.cxx
$
$ ./a.out
3 4 5 6 <ctrl-d>
The sum is: 18
```

- i dati sono letti nel `while` dall'istruzione `std::cin >>value`
- il **valore di ritorno** viene verificato dal `while` : se il valore di ritorno è valido, il risultato è `true`; in caso contrario (es: valore non intero, `end-of-file`) il risultato è `false`

end-of-file: UNIX <ctrl-d>, WINDOWS <ctrl-z>

# L'Istruzione Condizionale `if`

- Ne esistono due forme

```
if (espressione) statement
if (espressione) statement_1 else statement_2
```

- A multiway conditional statement

```
if (espressione_1)
    statement_1
else if (espressione_2)
    statement_2

...

else
    statement_n
```

# Esempio di Utilizzo

```
// calcolatore.cxx - Esegue semplici calcoli
#include <iostream>
int main( )
{
    using namespace std;
    cout << "Introduci una espressione (numero op numero) :";
    double n1, n2;
    char op;
    cin >> n1 >> op >> n2;
    double val;
    if (op == '+')    val = n1 + n2;
    else if (op == '-') val = n1 - n2;
    else if (op == '/') val = n1 / n2;
    else if (op == '*') val = n1 * n2;
    else {
        cout << "Operatore \"" << op << "\" sconosciuto \n";
        return 1;
    }
    cout << n1 << op << n2 << " = " << val << endl;
    return 0;
}
```

```
Introduci una espressione (numero op numero) : 1 + 1
1 + 1 = 2
```



: In C++ '=' indica assegnazioni e '==' uguaglianze



# Formattazione del Codice

Alcuni linguaggi sono line-oriented (es. FORTRAN)

- C, C++: il punto-e-virgola, ' ; ' **marca la fine di una istruzione**
- L'utilizzo di **spaziatori** (<tab>, <space>) e ' \n ' **è libera**
- Unica regola: **le parole chiave (tokens) non possono essere separate**

```
// myfirst.cpp - stampa dei messaggi sullo schermo
```

```
#include <iostream>
int main( )
{
    using namespace std;
    cout << "Il nostro primo programma.";
    cout << endl;
    cout << "Impariamo il C++" << endl;
    return 0;
}
```

Codice ben  
formattato

# Codice Mal Formattato

Il codice seguente è valido ma [difficile da leggere](#) ...

```
// myfirst.cpp - stampa dei messaggi sullo schermo

#include <iostream>
int
main
( ) { using namespace std
;
    cout <<
    "Il nostro primo programma."
;
    cout << endl; cout << "Impariamo il C++"
<<
    endl; return
    0
;
}
```

Codice mal  
formattato

# Consigli sulla Formattazione

- C++ is a **free-format language**, → dove mettiamo le parentesi graffe, l'indentazione, commenti, dove andiamo a capo non hanno effetti sul programma
- le scelte che facciamo hanno una ripercussione sulla **leggibilità del codice**
- **non esiste un unico stile corretto**, ma una volta scelto, siate **coerenti e consistenti** durante la scrittura del codice
- Alcuni suggerimenti:
  1. Mantenere **una sola istruzione per linea**
  2. Le **parentesi di apertura e chiusura** del blocco di una funzione vanno posti **su righe separate**
  3. Tutte le **istruzioni all'interno di un blocco** vanno **indentate** rispetto alle parentesi
- Alcune referenze alla letteratura sugli standard di codifica in rete:
- **GOOGLE C++ style guide** : <http://goo.gl/iFKPU8>
- **Linux kernel coding style** : <http://goo.gl/kyCGtj>
- **Wikipedia indent style** : <http://goo.gl/jPLr>